Practice Midterm 3 Solutions

This page contains solutions to Practice Midterm 3.

Q1) C++ fundamentals and ADTs

```
one:    {0}
fish:   {1, 3, 5, 7, 10}
two:    {2, 8}
red:    {4, 9}
blue:   {6}
```

```
conc[toLowerCase(word)].add(pos++);
```

```
int findPhrase(string phrase, Map<string,Set<int>>& conc) {
    Vector<string> words = stringSplit(toLowerCase(phrase), " ");
    for (int start: conc[words[0]]) {   // foreach possible start pos
        int i = 1;
        // iterate over words in sequence and match each position
        while (i < words.size() && conc[words[i]].contains(start + i)) {
            i++;
        }
        if (i == words.size()) { // found entire phrase
            return start;
        }
    }
    return -1;
}
```

Q2) Code study: ADTs and Big-O

echoVector	O(N^2)
echoStack	O(N)
echoQueue	O(N)
echoSet	O(NlogN)

echoVector

```
v = {1, 3, 2, 5, 3, 6}
```

echoStack

```
s = {3, 5, 6, 3, 1, 0}
```

echoQueue

```
q = {3, 1, 5, 2, 6, 3}
```

echoSet

```
set = {1, 2, 3, 5, 6}
```

Queue iterate: The changed code goes into an infinite loop when echoing any non-empty queue.

Set iterate: An attempt to add/remove elements while iterating over a collection in for-each loop raises a runtime error.

Q3) Recursion

```
// return value is the summed area of white rectangles
double recMondrian(double x, double y, double w, double h) {
    if (w * h < 500) {
        drawRect(x, y, w, h, "grayscale");
        return 0;
    }
    int choice = randomInteger(1, 3);
    if (choice == 1) {              // split: none
        drawRect(x, y, w, h, "white");
        return w * h;
    } else if (choice == 2) {       // split: horiz
        return recMondrian(x, y - h/2, w, h/2) + recMondrian(x, y, w, h/2);
    } else {                        // split: vert
        return recMondrian(x, y, w/2, h) + recMondrian(x + w/2, y, w/2, h);
    }
}

double drawMondrianRect(double x, double y, double w, double h) {
    double white = recMondrian(x, y, w, h);
    return (100 * white)/(w * h);
}
```

Q4) Recursive backtracking

Start with template provided by **`printTotalSpent`** and make these changes:

1. backtrack: stop at first success, prune dead ends, return true/false
2. allow choose zero-one-up to N
3. decrement inventory on success
4. track state using vector index, no copies/edits to vector

There are several possible approaches that can work. The solution below is the one that is most similar to the template.

```
bool spendAll(double amount, Vector<itemT>& inventory, int index) {
    if (amount == 0) {                              // success
        return true;
    }
    if (amout < 0 || index == inventory.size()) { // failure
        return false;
    }
    itemT cur = inventory[index];
    for (int n = 0; n <= cur.count; n++) { // n is number of item put in
basket
        inventory[index].count -= n;                                  //
choose
        if (spendAll(amount - n*cur.price, inventory, index + 1)) { //
explore
            return true; // stop at first success
        }
        inventory[index].count += n;                                 //
unchoose
    }
    return false;
}

bool spendAll(double amount, Vector<itemT>& inventory) {
    return spendAll(amount, inventory, 0);
}
```


Website programming by Julie Zelenski with modifications by Sean Szumlanski • Styles adapted from Chris Piech • This page last updated 2025-Apr-21